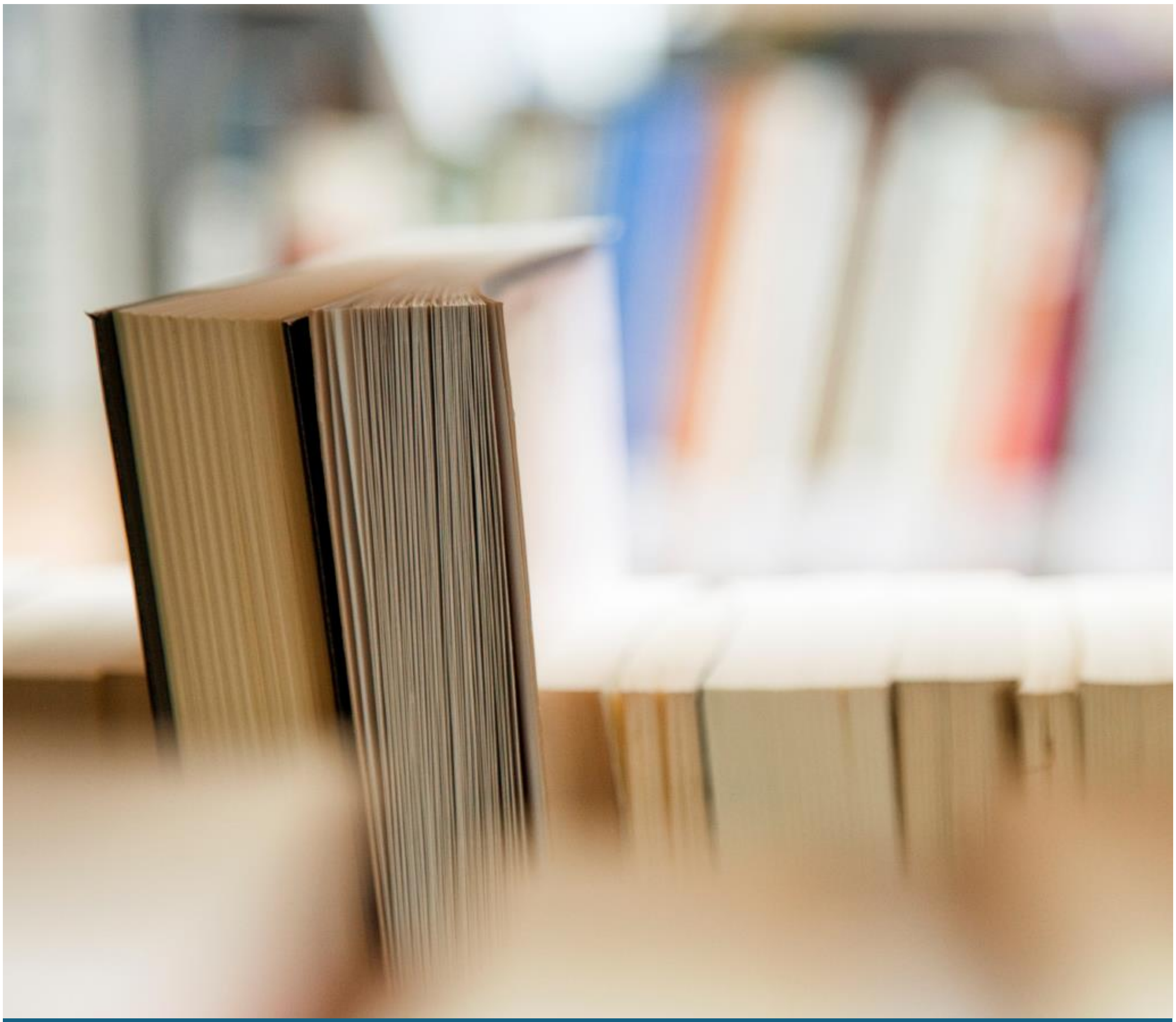


pLLM Final Report

Under the Guidance of Prof. David Schlangen

Akash, Shahrukh & Swapnil 9/29/24 PM: Large Language Models

Table of Contents

Abbreviation	Full form
GINI	Guidance on Intelligent Nutritional Intake
UI	User Interface
UX	User Experience
GPT	Generative Pre-trained Transformers
API	Application Programming Interface
LLM	Large Language Model
ML	Machine Learning
AI	Artificial Intelligence
OS	Operating System
USDA	U.S Department of Agriculture
SSL	Secure Sockets Layer
TSL	Transfer Support Layer
CI	Continuous Integration
CD	Continuous Development
MVP	Minimal Viable Product
SDK	Software Development Kit

1. Introduction

GINI is a nutritional chatbot that processes the input as language passes it through its GPT-3 pre-trained model and produces outputs in the form of text for its users.

1.1 Problem Objective

Gini is a personalised nutrition chatbot. It aims to provide its users with three main features: Tailored meal planning, recipe suggestions, and detailed nutritional information. The chatbot molds its behaviour based on its users' individual dietary needs and preferences.

1.2 Problem Statement

Gini addresses the common challenges in nutrition. These challenges are difficulty in meal planning, limited recipe options, and lack of nutritional information. The system is not only helping its users eat better but is also educating them in their daily lives.

1.3 Target Audience

Gini is a publicly, accessible to all system but the target audience of the system is:

I. Fitness enthusiasts with specific dietary goals (e.g., muscle gain, weight loss).
II. Individuals with food allergies or dietary restrictions (e.g., gluten-free, vegan).
III. People seeking to adopt healthier eating habits.
IV. Users who want to improve their knowledge on the topic of nutrition.

1.4 Goals and Deliverables

The goals and deliverables of the system were structured as follows:

I. Building an MVP that offers personalized meal planning, recipe suggestions, and nutritional breakdown.
II. Integrate a feedback mechanism or question chaining with memory to improve further recommendations.
III. Aim for API integration for enhanced nutrient data (e.g., USDA, Spoonacular).

2. System Overview

2.1 Architecture Diagram

The following figure illustrates the architecture. It shows the interaction between the user interface, LLM backend and LangChain (OpenAI API).

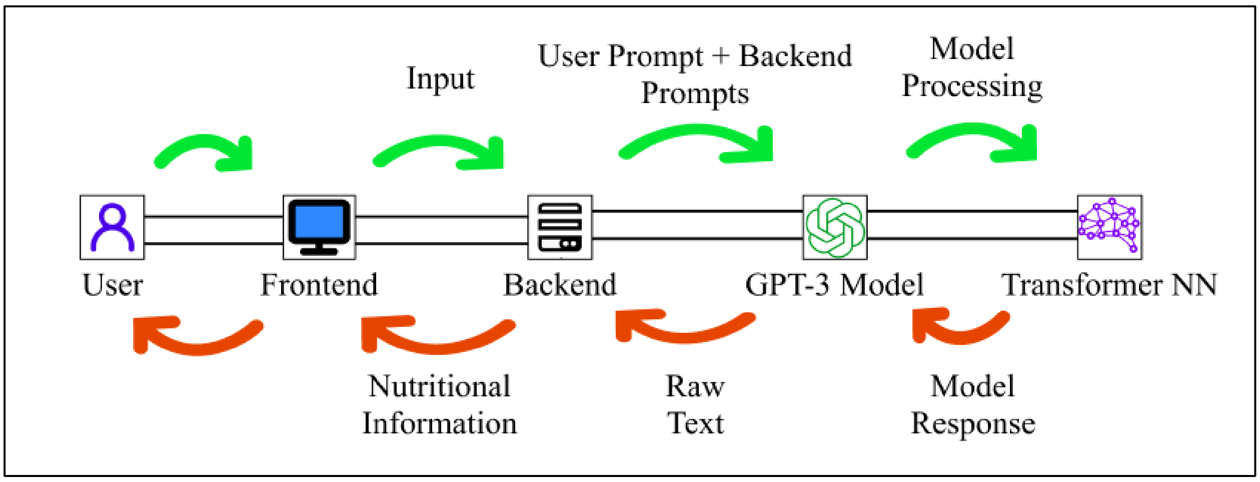


Figure 1: System Architecture

2.2 Technology Stack

The following technologies were used in the development of the system:

I. **Frontend**: Streamlit (Python library for frontend).
II. **Backend**: Python + LangChain with OpenAI's GPT-3.5 Turbo for natural language processing and response generation.
III. **Deployment**: Streamlit Community Cloud.

2.3 Workflow Overview

The workflow starts with user input (preferences, dietary restrictions), followed by prompt selection based on input type, and output generation (meal plans, recipes, nutrient breakdown) using multi-prompt chains.

I. **User Input**: The user enters a query through a text box.
II. **Router Chain**: The input is routed to the most appropriate prompt based on its content (e.g., recipe suggestion, meal plan).
III. **Prompt Processing**: The selected chain processes the query using the GPT-3.5 model and the corresponding prompt template.
IV. **Response Display**: The chatbot's response (meal plan, recipe, nutrient breakdown) is displayed on the Streamlit interface.
V. **Further Queries**: Users can ask follow-up questions, which the system processes similarly.

2.4 Tools and Libraries Used

Libraries:

I. **LangChain**:

a. Used for building chains of prompts and managing the routing between different prompt templates.

b. Integration of the LLM (language model) with custom templates for generating dynamic responses.

c. Submodules being used include:

   i. MultiPromptChain

   ii. LLMRouterChain

   iii. RouterOutputParser

   iv. LLMChain

II. **Streamlit**:

a. Provides the user interface (UI) for the chatbot. It allows us to create an interactive web-based frontend where users can enter queries, view responses, and interact with the system.

b. Key Streamlit components used:

   i. st.text_area: To capture user queries.

   ii. st.button: To handle user input and trigger responses.

   iii. st.markdown: To display formatted chatbot responses.

III. **OpenAI API (GPT-3.5-turbo)**:

a. The language model (LLM) is used for generating personalized responses based on user input.

b. Integrated via the ChatOpenAI class from LangChain.

IV. **IPython Display**:

a. Specifically using Markdown from the IPython library to display Markdown content. Although this is typically used in Jupyter notebooks, the use in this script is likely geared toward rendering output in Markdown format.

Tools:

I. **Python's os Module**:

- Used to securely retrieve the OpenAI API key from environment variables.

3. Gini's Approaches and the Final Model being used

Gini had some iterations after which it was decided by the team to use a pre-trained model like GPT-3.5 Turbo to obtain our results. The following few headings will try to cover the facts that led to these decisions during the development of this project.

3.1 Initial Architecture of Gini: A "Bare-bones Approach"

A fine-tuned **Transformer-based model** (BERT) was trained on a custom dataset of recipes, nutrients, and dietary restrictions. It classified the user query, generated texts (e.g., meal plans), and provided structured outputs like nutritional data. However, the team faced an issue with the accuracy and coherence of the outputs produced. This was even though multiple hyperparameters were tested and tried.

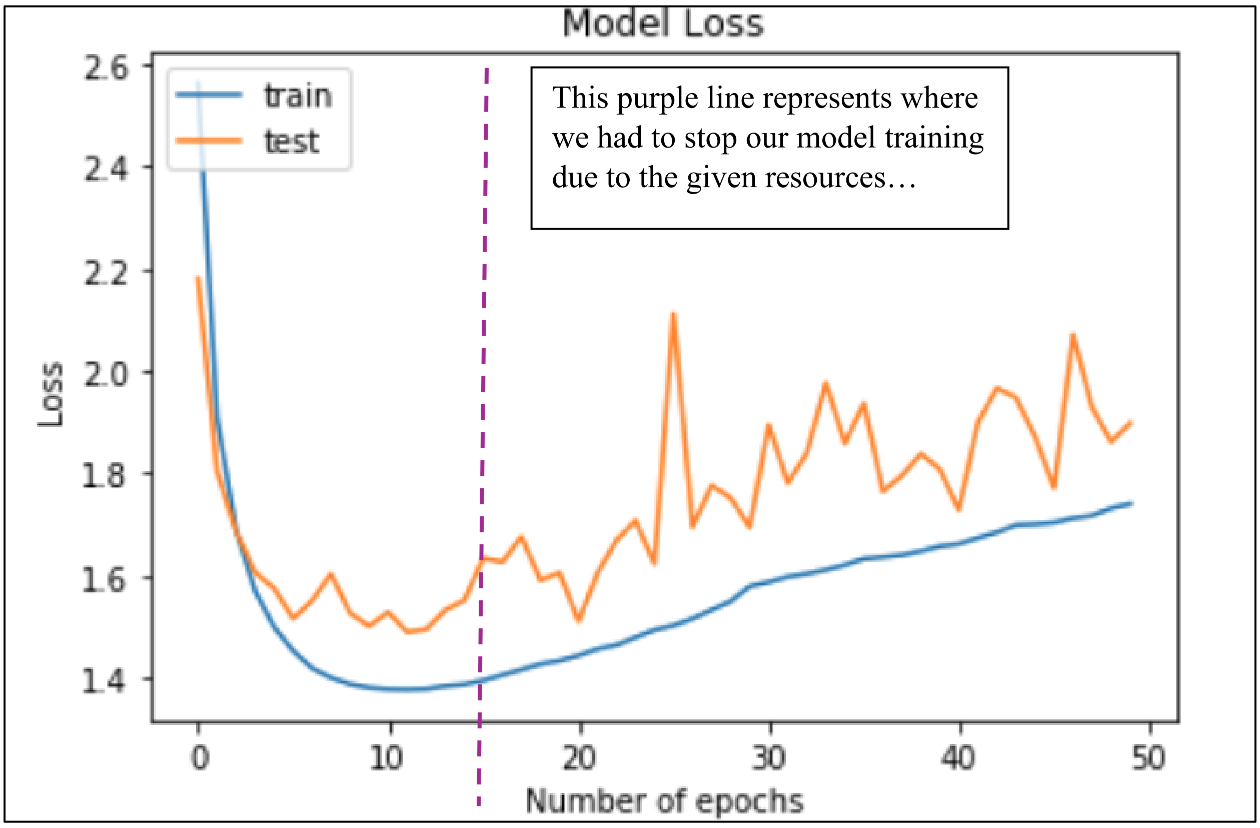


Figure 2: Loss VS Number of Epochs

In the end, the team had to give up due to a limitation of resources and bad model performance. The team decided to use a pre-trained model and utilize its powers which proved to be very convincing in comparison to the "Bare-bones approach".

3.2 Current Approach Being Used by GINI

The current approach of **Gini**, which uses **GPT-3.5-turbo**, offers a sophisticated way to generate highly personalized meal plans, recipe suggestions, and nutritional advice. Here are some key insights into how Gini utilizes GPT-3.5-turbo to achieve its goals, producing coherent and convincing results:

1. **Leveraging GPT-3.5-turbo's Natural Language Understanding**

- **GPT-3.5-turbo** is highly proficient in understanding and generating natural language, making it an ideal model for conversationally interacting with users. Gini uses this strength to interpret user inputs, whether they are direct questions ("What should I eat for dinner?") or more complex dietary requirements ("I need a high-protein, vegan meal plan").
- **Insight**: GPT-3.5's ability to parse complex sentences, identify key entities (like ingredients or dietary restrictions), and generate contextually relevant responses ensures that Gini delivers **tailored meal suggestions** in a way that feels natural to the user.

2. Multi-Prompt Chain System for Specialized Queries

- Gini implements a **multi-prompt chain system** using LangChain, where each type of query (e.g., meal planning, recipe suggestions, nutrient breakdowns) is routed to the appropriate **prompt template**. This allows Gini to generate focused and relevant responses based on user input.
- **Insight**: By dividing user queries into specialized prompts, Gini ensures that the output is relevant and accurate. For instance, a user asking for a "balanced meal plan" is routed to a specific prompt that focuses on generating a full-day or week-long plan, while another user asking for a "nutrient breakdown" is routed to a prompt designed to provide detailed macronutrient and micronutrient information.

3. Prompt Engineering for Personalization

- Gini uses **carefully designed prompt templates** for each core functionality (recipe suggestions, nutrient breakdown, dietary advice). These templates are written to leverage GPT-3.5's ability to adapt its language generation based on the instructions it receives.
- **Insight**: Gini achieves personalization by embedding user-specific details directly into the prompts. For example, if a user prefers "high-protein meals," the prompt will include that dietary restriction, allowing GPT-3.5-turbo to generate suggestions that align with the user's goals.

Sample Prompt:

"You are an expert nutritionist. Provide a meal plan for a person who follows a high-protein vegan diet. Include recipes and a detailed nutrient breakdown for each meal."

4. Real-Time Response Generation

- GPT-3.5-turbo is a pre-trained model that allows for **real-time generation of responses** based on user input. Gini harnesses this capability to provide instant feedback, whether it's generating a recipe or offering nutritional advice, without the need for predefined datasets or rigid rules.
- **Insight**: By relying on GPT-3.5-turbo's generative capabilities, Gini can provide **dynamic, on-the-fly meal suggestions** that feel fresh and customized. The model's ability to generate coherent, logical steps (e.g., in recipe instructions) adds to the user experience, making the interaction feel conversational yet informative.

5. **Handling Ambiguity and Complex Queries**

- One of GPT-3.5's strengths is its ability to handle **ambiguous or open-ended queries**. Gini leverages this by allowing users to input general requests, and GPT-3.5-turbo responds with relevant options, often guiding the user by providing clarifying follow-up suggestions (e.g., "Would you prefer a vegetarian or vegan meal?").
- **Insight**: This flexibility allows Gini to handle **vague queries** like "I want something healthy" and return suggestions that make sense in context. The ability to interpret unclear inputs and provide sensible clarifications enhances user satisfaction.

6. **Context Awareness and Conversational Flow**

- GPT-3.5-turbo can maintain **context across multiple turns** in a conversation, which is important for Gini's interaction style. If a user asks for a meal plan and then follows up with specific dietary restrictions or preferences, the model can integrate that new information into the ongoing conversation.
- **Insight**: This context-awareness gives Gini the ability to provide a **fluid conversational experience**, where follow-up questions or modifications to meal plans feel natural and consistent with prior interactions.

7. **Scalability and Adaptability**

- Since GPT-3.5-turbo is a pre-trained model with massive knowledge across domains, Gini can easily scale and adapt to new use cases without needing to retrain the model. This allows Gini to cover a broad range of dietary needs and continually expand its feature set (e.g., integrating fitness-based meal plans or more complex dietary restrictions) by simply updating the prompt templates.
- **Insight**: This scalability allows Gini to evolve without significant re-engineering efforts, making it highly adaptable to future needs, such as adding **API integrations** for real-time nutritional data or **user feedback loops**.

8. **Ensuring Coherence and Persuasiveness**

- The quality of responses generated by GPT-3.5-turbo is consistently high in terms of **coherence and logical flow**, especially when handling complex tasks like suggesting recipes with step-by-step instructions or providing a nutrient breakdown for a full-day meal plan.
- **Insight**: By relying on GPT-3.5's ability to produce **convincing and structured text**, Gini provides users with not just informative responses, but ones that feel professionally curated. This adds to the trustworthiness and usability of the system, especially for users seeking reliable nutritional advice.

9. **Future Expandability with External APIs**

- While Gini currently relies on GPT-3.5-turbo, the system has been designed with future scalability in mind. Integrating **external APIs** for live nutritional data (such as **USDA FoodData Central**) will allow Gini to pull real-time nutritional updates and ensure that the meal plans and advice remain up to date.

- **Insight**: GPT-3.5-turbo provides a robust backbone for handling general language generation tasks, but the planned integration of external APIs will further enhance Gini’s accuracy and relevance, particularly in cases where nutritional data is subject to frequent updates.

4. Core Components

4.1 Recipe Suggestions

Gini offers personalized and healthy recipe suggestions based on the ingredients provided by the user. By leveraging the **RecipeSuggestions_template**, the system can generate recipes that include step-by-step preparation instructions, cooking times, and a detailed nutrient breakdown for each serving. The system ensures that the suggested recipes align with the user's dietary preferences, such as vegetarian, gluten-free, or high-protein meals. Gini's recipe suggestions go beyond basic meals, emphasizing balanced nutrition to meet the user's goals.

Key points:

I. User inputs a list of ingredients.
II. The `RecipeSuggestions_template` is selected by the router chain based on the input.
III. The OpenAI model generates a list of recipes that include cooking instructions, preparation time, and a breakdown of calories, proteins, fats, carbohydrates, vitamins, and minerals.

4.2 Nutrient Breakdown

Gini provides detailed nutrient breakdowns for food items using the `NutrientBreakdown_template`. The system delivers essential information, including macronutrients (proteins, fats, carbohydrates) and micronutrients (vitamins and minerals), in a clear and concise format. This feature helps users understand the nutritional value of individual ingredients or meals, aiding them in making informed dietary decisions. The nutrient breakdown feature is designed for flexibility and can process simple queries like a single food item or complex meal compositions.

Key points:

I. User inputs a food item.
II. The `NutrientBreakdown_template` triggers the LLM to generate a detailed nutrient breakdown.
III. The system outputs nutritional information including calorie content, macronutrients (protein, fats, carbs), and micronutrients (vitamins, minerals).

4.3 Meal Plan generation

Gini generates personalized meal plans based on user dietary preferences and goals (such as weight loss, muscle gain, or maintaining a balanced diet). The `MealPlan_template` dynamically creates meal plans that cover an entire day or week, depending on the user's request. Each meal is accompanied by a detailed nutrient breakdown to ensure that it aligns with the user's health objectives. Gini considers the user's input preferences (e.g., vegetarian, high-protein) and provides meal plans that are nutritionally balanced and customized to support specific dietary goals.

Key points:

I. User inputs dietary preferences and goals (e.g., weight loss, muscle gain).

II. The `MealPlan_template` is invoked, which uses the LLM to generate a personalized meal plan.
III. The system provides a nutrient breakdown for each meal, ensuring that users meet their caloric and macronutrient requirements.

4.4 Exercise And Nutrition Alignment

This component focuses on aligning nutrition with the user's exercise routine, providing detailed pre- and post-workout meal suggestions. Using the `ExerciseNutrition_template`, Gini tailors meals based on the user's exercise type, duration, and intensity. The system considers nutrient timing, such as optimizing carbohydrate intake pre-workout for energy and protein intake post-workout for muscle recovery. This integration allows Gini to offer targeted nutrition advice that supports fitness goals.

Key points:

I. Users input their exercise routine, including details like workout duration and intensity.
II. The `ExerciseNutrition_template` processes this input and generates tailored meal recommendations.
III. Nutrient timing suggestions are provided to ensure optimal energy and recovery.

4.5 Dietary Restrictions Handling

Gini addresses dietary restrictions by providing safe food choices and recipes that cater to specific needs, such as gluten intolerance, veganism, or dairy allergies. The DietaryRestrictions_template ensures that suggested meals and recipes adhere to the user's restrictions while still offering balanced nutrition. The system also recommends substitutions for common allergens and provides creative alternatives to meet dietary requirements without sacrificing nutritional value.

Key points:

I. User inputs their dietary restrictions (e.g., gluten-free, vegan).
II. The DietaryRestrictions_template is selected and processes the input.
III. Gini generates a list of allowed foods, avoids restricted items, and provides recipes that adhere to the restrictions.

4.6 Weight Management

Gini assists users in managing their weight by generating meal plans and offering tips for weight loss, gain, or maintenance. The `WeightManagement_template` is tailored to provide users with comprehensive advice on caloric intake, portion control, and nutrient distribution. Whether the user's goal is to lose fat or gain muscle, Gini ensures that the recommended meal plans align with their specific caloric and macronutrient needs.

Key points:

I. User specifies a goal (weight loss, gain, or maintenance).

II. The `WeightManagement_template` processes this goal and generates meal plans and tips.
III. The system includes advice on portion control, caloric deficit or surplus, and nutrient distribution to help users reach their desired weight.

4.7 Food Allergy Management

Gini helps users manage food allergies by identifying common allergens and recommending safe alternatives. Using the `FoodAllergies_template`, the system provides users with a list of foods to avoid, based on their specific allergies, and suggests safe recipes that eliminate these allergens. This component ensures that users receive safe, allergy-free meal recommendations without compromising on nutrition.

Key points:

I. User inputs their specific food allergies.
II. The `FoodAllergies_template` triggers the system to identify allergens and safe alternatives.
III. Gini generates allergy-friendly recipes and suggests safe foods while avoiding common allergens.

4.8 Digestive Health Guidance

Gini provides recommendations for maintaining digestive health using the `DigestiveHealth_template`. The system suggests foods that promote gut health, such as high-fibre options, and offers advice on managing common digestive issues like bloating or indigestion. Users receive practical tips on habits and foods that can improve digestion, helping them maintain a healthy gut and overall well-being.

Key points:

I. User inputs concerns related to digestive health.
II. The `DigestiveHealth_template` processes the input and generates advice on improving digestion.
III. Gini suggests gut-friendly foods, habits, and tips to alleviate digestive discomfort.

4.9 Plant-based Diet Advice

Gini offers comprehensive guidance for users following a plant-based diet through the `PlantBasedDiet_template`. The system ensures that users receive adequate protein, vitamins, and minerals from plant sources and provides balanced meal suggestions that support their dietary preferences. Gini's meal plans help users achieve nutritional balance while adhering to a plant-based lifestyle.

Key points:

I. User inputs preferences for a plant-based diet.
II. The PlantBasedDiet_template generates balanced meal suggestions that focus on plant-based protein and micronutrient intake.
III. Gini offers meal plans that ensure users meet their nutritional requirements while following a plant-based lifestyle.

5. Implementation Details

5.1 Template Design for Prompts

Gini uses a modular approach for handling user queries through **custom-designed prompt templates**. Each template is carefully crafted to generate contextually accurate responses for different categories of queries such as recipe suggestions, nutrient breakdowns, meal plans, and dietary advice. The templates serve as structured instructions for the language model (GPT-3.5-turbo) to interpret user inputs and respond effectively.

Technical Breakdown:

I. Each prompt is embedded in a chain, with variable placeholders (e.g., {input}) that are dynamically filled based on user input.

II. For example, the `RecipeSuggestions_template` takes the user-provided ingredients and generates a list of recipes along with nutrient breakdowns, while the `MealPlan_template` processes dietary preferences and produces a full meal plan.

III. These prompt templates ensure that Gini can handle a wide variety of queries, ranging from simple nutrient requests to more complex meal-planning tasks.

5.2 Multi-Prompt Chain Setup

The **multi-prompt chain** system is integral to Gini's functionality, allowing for dynamic routing of user queries to the most appropriate prompt template. This architecture ensures that the system responds accurately based on the type of query, without overwhelming the language model with irrelevant details.

Technical Breakdown:

I. The **MultiPromptChain** is implemented by creating individual chains for each prompt template (e.g., `NutrientBreakdown_template`, `RecipeSuggestions_template`).

II. A router chain (LLMRouterChain) is configured to decide which prompt chain to activate based on the user's input.

III. The decision-making process is further refined by a **RouterOutputParser**, which modifies or adjusts the input if necessary before it is passed to the corresponding prompt template.

IV. This setup reduces errors and increases the specificity of responses, as each query is funnelled through a specialized template, ensuring accurate, context-aware output .

5.3 API Integration

Although not implemented in the current system, the code is designed with **future API integrations** in mind. The long-term goal includes integrating external APIs such as **USDA FoodData Central** or **Spoonacular** to provide real-time and highly accurate nutrient data.

Technical Breakdown:

I. The architecture can easily accommodate API calls within the existing prompt templates. For instance, during a nutrient breakdown request, the system could fetch the latest nutritional data via the USDA API.

II. This would enhance the system's ability to provide real-time updates on food items, offering more accurate and comprehensive nutritional information.

III. Future development plans include seamless API integration with Gini's backend, allowing the system to continuously update its food database and generate more precise meal plans.

5.4 Handling User Input

The system's approach to **handling user input** is both structured and dynamic, ensuring that users receive accurate responses tailored to their preferences. The system captures inputs such as dietary preferences, goals (e.g., muscle gain, weight loss), or dietary restrictions, and then routes them to the appropriate prompt via the router chain.

Technical Breakdown:

I. Inputs are captured through **Streamlit's text area widgets**, where users can describe their dietary preferences, ingredient lists, or health goals.

II. The input is then processed by the **multi-prompt chain system**, where the router determines the type of query (e.g., recipe suggestion, nutrient breakdown).

III. The system dynamically fills the placeholder variables in the prompt templates (e.g., {input}) with user-provided data and passes it to the LLM for generating the final response.

IV. The input handling mechanism is flexible, allowing the user to adjust queries based on feedback, which triggers the appropriate response modification.

5.5 Data Flow and Decision Routing

The **decision-routing system** plays a critical role in ensuring that each query is processed through the most relevant prompt chain, providing personalized and contextually relevant outputs.

Technical Breakdown:

I. When a user submits a query, it is first evaluated by the **Router Chain**. This chain is responsible for determining which prompt (e.g., `ExerciseNutrition_template`, `MealPlan_template`) is best suited for processing the input.

II. The router uses a predefined set of candidate prompts (available prompts listed as destinations), which are dynamically referenced in the multi-prompt chain.

III. Based on the decision made by the router, the system routes the input to the correct chain, which then engages the GPT-3.5 model to generate the desired output.

IV. The data flow ensures efficient processing of inputs, avoiding irrelevant prompts, and improving response accuracy through targeted routing.

5.6 Error Handling and Edge Cases

Error handling is essential to ensuring that Gini provides accurate responses even when faced with incomplete, ambiguous, or out-of-scope user inputs. The system includes mechanisms for managing such edge cases, prompting users for clarification or defaulting to a generic response when necessary.

Technical Breakdown:

I. The system incorporates an **input guard** to check if the prompt is relevant to nutrition or health. If the input is invalid, the system responds with "INVALID PROMPT," ensuring that irrelevant queries do not result in incorrect or confusing outputs.

II. In the case of incomplete inputs (e.g., missing ingredient details for a recipe suggestion), the system can prompt the user for more information, or default to a basic response based on the available data.

III. Vague inputs are handled by the router chain’s ability to adjust or revise the input before routing it to the LLM. For example, if the user’s query lacks context, the system will refine the query to better suit the selected prompt.

IV. By incorporating this feedback loop and clarification mechanism, the system ensures that users receive relevant, high-quality responses even when faced with ambiguous queries.

6. User Experience (UX) Design

6.1 User Interaction Flow

The interaction flow for Gini is centred around an intuitive chatbot interface that allows users to input dietary preferences, goals, and restrictions. Users can interact with Gini through a simple text input box where they describe their needs (e.g., "I want a high-protein meal plan for weight loss" or "I have gluten intolerance"). The system then processes the input through the **multi-prompt chain** and responds with appropriate meal plans, recipes, or nutrient breakdowns.

Technical Breakdown:

I. **Input**: Users type queries into a **Streamlit text area**. Common inputs include dietary restrictions, ingredients for recipes, or fitness goals.

II. **Output**: The chatbot dynamically generates responses based on the user's input, returning personalized meal plans, nutrient breakdowns, or other dietary advice.

III. **Interaction Loop**: After receiving the response, users can provide further input or adjust their queries for additional details, creating a loop of user engagement and personalized responses

6.2 UI/UX Design Decisions (Streamlit Interface)

The **Streamlit-based interface** for Gini is designed to be clean, minimalistic, and user-friendly. The simplicity of the design ensures that users can easily interact with the chatbot without requiring technical expertise. The focus is on providing immediate, clear, and personalized responses to user queries.

Technical Breakdown:

I. **Ease of Use**: The main UI component is a single **text input area** where users type their queries. The system offers immediate feedback once the user presses the "Submit" button.

II. **Interactive Features**: The design includes interactive elements like buttons to submit initial queries and follow-up questions. These buttons help control the flow of interaction.

III. **Feedback Mechanism**: Future iterations of the system plan to incorporate a **rating system** or thumbs-up/thumbs-down buttons to allow users to give feedback on the quality of the responses they receive. This feedback will be integrated into the system's backend for continuous improvement.

6.3 Accessibility Considerations

Gini's design adheres to accessibility guidelines to ensure the platform is usable by individuals with disabilities. Since Gini operates via a text-based chatbot interface, it is accessible for users who rely on screen readers or keyboard navigation. Additionally, text-based interaction provides an inclusive experience for users with hearing or visual impairments.

Technical Breakdown:

I. **Text-Based Input**: The primary interaction is text-based, allowing users to input queries using assistive technologies such as screen readers or text-to-speech software.

II. **Accessible Navigation**: The simple, linear structure of the interface (input box followed by response) is designed for ease of use across a wide range of devices and assistive technologies.

III. **Visual Simplicity**: The interface avoids overwhelming users with unnecessary visuals, focusing on clear, concise text responses that are easy to read and understand.

6.4 User Feedback and Validation

User feedback is an integral part of Gini’s development process. Feedback is collected from users after they interact with the system to gauge satisfaction with the meal plans, recipes, or nutritional advice provided. This feedback is used to refine Gini's prompts, improving personalization and accuracy over time.

Technical Breakdown:

I. **Feedback Collection**: While the current implementation does not include direct feedback mechanisms, the **planned feedback system** will use interactive features (like rating or thumbs-up/down buttons) to gather real-time user feedback on the quality of the responses.

II. **Continuous Improvement**: Collected feedback will be fed back into the system to adjust the prompt templates and decision-routing mechanisms, enabling Gini to evolve and become more personalized over time. This continuous feedback loop ensures that the system remains relevant and adapts to user preferences.

III. **Usability Testing**: The system undergoes regular testing with users to identify pain points or areas for improvement, ensuring that Gini remains intuitive and useful for a diverse audience

7. Evaluation and Testing

7.1 Functional Testing of Components

Functional testing was carried out to ensure that Gini’s core components—**meal planning**, **recipe suggestions**, and **nutrient breakdown**—performed correctly and met the system's functional requirements. The aim was to verify that the system could handle various input scenarios and generate relevant outputs that aligned with the user’s dietary goals.

Technical Breakdown:

Recipe Suggestions:

a. Tested by inputting different combinations of ingredients and verifying the system’s ability to retrieve recipes that matched the user’s input. The test ensured that recipes included detailed preparation steps, cooking times, and accurate nutritional breakdowns.

Meal Plans:

b. Evaluated based on user dietary preferences (e.g., vegan, gluten-free, high-protein). Each test ensured the generated meal plans were nutritionally balanced and aligned with the dietary goals specified by the user, such as weight loss, maintenance, or muscle gain.

Nutrient Breakdown:

c. The accuracy of the nutrient breakdown feature was validated by comparing the system’s output with known food databases and nutritional resources. Each test ensured that the system provided precise macronutrient and micronutrient data for individual food items or meals.

7.2 Unit Testing Approach

The system's **unit testing** was performed manually to validate the behaviour of individual prompt templates and ensure that they functioned correctly when handling different types of user inputs. Each prompt chain (e.g., recipe suggestion, nutrient breakdown) was tested to verify the correctness of the response generation process.

Technical Breakdown:

I. **Prompt Templates**: Each template (e.g., MealPlan_template, RecipeSuggestions_template, NutrientBreakdown_template) was tested individually to ensure it processed inputs as expected and returned coherent, relevant responses.

II. **Edge Case Testing**: Inputs that were vague, incomplete, or ambiguous were tested to ensure that the system could either provide a reasonable response or prompt the user for further clarification. This testing ensured that Gini handled unexpected inputs gracefully.

III. **Router Chain**: The decision-routing mechanism was tested to verify that the correct prompt template was invoked based on the user's input type (e.g., meal plan generation vs. nutrient breakdown).

7.3 User Testing and Feedback

User acceptance testing (UAT) was conducted to assess Gini's performance from an end-user perspective. A group of test users interacted with Gini by submitting various queries (e.g., dietary preferences, meal planning requests, and nutrient data queries), and feedback was gathered to assess how well the system met their needs.

Technical Breakdown:

I. **User Feedback**: Users provided feedback on the accuracy and relevance of Gini's responses. This feedback helped identify areas for improvement, such as refining meal plan suggestions for complex dietary restrictions or adjusting the nutrient breakdown feature for specific food items.

II. **Feature Refinement**: Based on user feedback, several features were modified to improve user satisfaction. For instance, Gini's handling of dietary restrictions was enhanced by refining the decision-routing system and clarifying user prompts to prevent misunderstandings.

7.4 Performance Evaluation

Performance testing focused on ensuring that Gini responded to user queries within an acceptable timeframe and could handle multiple simultaneous requests without a significant degradation in performance.

Technical Breakdown:

I. **Response Time**: Tests were conducted to measure Gini's response time under different load conditions (e.g., multiple concurrent users, complex meal plan generation). The system's backend was designed to handle varying levels of user traffic while maintaining responsive interactions.

II. **Scalability**: The system's backend, deployed on cloud platforms (e.g., **Render.com**), was evaluated for its ability to scale effectively. Load testing was performed to ensure that Gini could handle an increased number of users without crashing or slowing down significantly.

7.5 Accuracy of Nutrient Data

To ensure that Gini's nutrient breakdown feature provided reliable and accurate information, the nutrient data generated by the system was compared against known nutritional databases such as **USDA FoodData Central**.

Technical Breakdown:

I. **Data Validation**: The nutritional data (e.g., calorie count, macronutrient content) generated for various foods and recipes were cross-referenced with reliable databases and resources to verify accuracy. This helped ensure that users could trust the information provided by Gini.

II. **Continuous Improvement**: As part of the development roadmap, future iterations of Gini will include API integrations with external databases like USDA to further enhance the accuracy and comprehensiveness of the nutrient data.

8. Self-Assessments

8.1 Strengths of the System

Gini demonstrates significant strengths in its ability to generate personalized, flexible meal plans tailored to user preferences and dietary goals. The system utilizes **custom prompt templates** to dynamically adjust to user inputs, whether for recipes, meal plans, or nutritional advice. Its flexibility allows users to receive meal plans that match specific requirements, such as vegan, gluten-free, or high-protein diets, all while providing accurate nutritional breakdowns.

Another key strength lies in its **user feedback mechanism**. Although not yet fully implemented, Gini has been designed to incorporate user feedback into future iterations, allowing for continuous refinement of responses and meal plan suggestions. This adaptability will help Gini improve over time by learning from user preferences and interactions.

Key Strengths:

I. Dynamic and personalized meal planning tailored to user inputs.

II. Flexible and customizable prompt templates.

III. Incorporation of user feedback mechanisms for continuous improvement.

8.2 Limitations and Caveats

Despite its strengths, Gini has certain limitations that affect its overall functionality. The **lack of memory** across user sessions is a key constraint. This means that Gini cannot retain user preferences (such as dietary restrictions or favourite ingredients) between sessions, limiting its ability to offer continuous personalization over time. This limitation impacts the user experience, especially for individuals who have specific, recurring dietary needs.

Another limitation is the system's reliance on **predefined prompt templates**. While the multi-prompt chain system is effective in routing inputs to the correct template, it lacks the flexibility to handle more nuanced or complex queries that fall outside the predefined templates. As a result, the system may fail to provide detailed, context-specific responses in cases where the user’s input is ambiguous or requires a higher level of understanding.

Key Limitations:

I. No persistent memory for storing user preferences across sessions.

II. Reliance on predefined prompt templates, limiting flexibility with complex queries .

8.3 Efficiency and Scalability

Gini performs efficiently for **single-user interactions** and provides quick, accurate responses when handling relatively simple queries. The current implementation, which leverages **LangChain** and the **OpenAI GPT-3.5 model**, is well-optimized for processing individual requests and returning personalized recommendations promptly.

However, scalability could become an issue when handling **larger datasets** or **multiple concurrent users**. While the system works effectively for small-scale operations, it may require further optimization to handle higher traffic or more complex interactions without performance degradation. Improvements to backend architecture and response-time optimizations will be required to support **real-time, multi-user scalability.**

Key Efficiency and Scalability Considerations:

I. Optimized for single-user interactions with fast, personalized responses.
II. Potential challenges in scaling to larger datasets or handling concurrent users without performance impacts.

8.4 Improvement Areas

Several areas of improvement have been identified to enhance Gini's overall functionality and user experience. A major improvement would be the introduction of **persistent user profiles**, which would allow the system to remember user preferences across sessions. This would enable Gini to provide even more tailored recommendations by leveraging user history to refine meal plans and suggestions over time.

Additionally, expanding the system's capability to handle more **complex dietary restrictions** would further increase its usefulness. Currently, Gini performs well with common dietary preferences like veganism or gluten-free diets, but more complex scenarios—such as users with multiple, overlapping restrictions (e.g., vegan, gluten-free, and low-carb)—require more nuanced handling. Expanding the range of dietary considerations would make Gini a more robust and inclusive tool.

Key Improvement Areas:

I. Implementing **persistent user profiles** for storing preferences and improving personalization.

II. Expanding the handling of **complex dietary restrictions** to cover a wider range of user needs.

9. Security and Privacy

9.1 Data Privacy Considerations

Gini’s current implementation does not store user data persistently, which inherently reduces the risk of sensitive information being exposed. The system processes user queries and dietary preferences in real time, meaning no long-term storage of personal information occurs. This **stateless design** helps ensure that user data is handled only during the active session, with no lingering data once the interaction ends.

Technical Breakdown:

I. **Real-time Data Processing**: Gini processes user input dynamically without storing it in a database, meaning all interactions are ephemeral and discarded after the response is generated.

II. **No Persistent Storage**: Since the system does not store user data (e.g., preferences, meal history), it reduces potential vulnerabilities associated with database breaches or unauthorized access to stored information.

III. **Limited Data Retention**: The lack of data retention limits privacy risks, as the system only processes and uses data during the active session without saving it for future use.

9.2 User Data Handling

Given that Gini does not rely on a database or any long-term storage mechanism, user data is handled on a **per-session basis**. The system ensures that all user input, such as dietary preferences and goals, is processed in memory and discarded after generating the required output. This **transient handling** of data reduces potential security risks, as no personal information is stored after the interaction.

Technical Breakdown:

I. **In-memory Processing**: User inputs are handled entirely in memory for the duration of the session, ensuring no long-term data retention.

II. **Data Lifecycle**: Once a user request (e.g., meal plan generation) is processed and the output is delivered, the input data is no longer stored or retained in the system.

III. **Data Security in Transit**: While Gini does not store user data, it is still essential to protect data during transmission. Encryption protocols (e.g., SSL/TLS) should be considered for secure communication between the front end (Streamlit) and the back end (LangChain + OpenAI API).

9.3 Potential Risks and Mitigations

Despite the stateless nature of Gini, there are still potential risks that need to be addressed, particularly regarding the security of data during active sessions and communication between the client (user interface) and the server (backend).

Potential Risks:

I. **Data Interception**: During an active session, the user's input could potentially be intercepted during transmission if not properly encrypted.

II. **Real-time Exposure**: While the system doesn't store data, there's still a risk of exposure if the transmission between the user interface (Streamlit) and the backend (OpenAI API) is not secured.

III. **API Security**: As Gini relies on external APIs (e.g., OpenAI's GPT-3.5), securing API keys and preventing unauthorized access to the backend is essential.

Mitigation Strategies:

I. **SSL/TLS Encryption**: To mitigate the risk of data interception, secure communication protocols such as SSL/TLS should be implemented to encrypt data in transit between the front and backend, ensuring that user inputs (e.g., dietary preferences) are protected during the session.

II. **Secure API Key Management**: The OpenAI API keys are securely handled using environment variables (os.environ.get("SECRET")) to prevent exposure in the codebase, reducing the risk of unauthorized access to Gini's backend services.

III. **Session Expiry**: Implementing a session timeout mechanism ensures that if a user is inactive for a set period, the session ends, further minimizing the risk of data exposure during inactive sessions.

10. Stretch Goals and Future Enhancements

10.1 User Preference Memory

A critical enhancement to Gini would be the introduction of **user preference memory**, enabling the system to store user data across sessions. This feature would allow Gini to deliver a more personalized and consistent experience by remembering dietary preferences, goals, and restrictions, even after the session ends.

Technical Breakdown:

I. **Persistent User Profiles**: By integrating a secure database (or cloud storage solution), Gini could store user profiles containing dietary preferences (e.g., vegan, gluten-free), favourite ingredients, and recurring goals (e.g., weight loss, muscle gain). This would allow the system to offer more personalized suggestions over time.

II. **Session-Based Personalization**: Gini would be able to access stored preferences and adjust recommendations dynamically based on user history, making meal planning and recipe suggestions more tailored to individual needs.

III. **Privacy Considerations**: Storing user data across sessions would require robust encryption and privacy protocols to protect sensitive information.

10.2 Dynamic API Integration (e.g., USDA)

Another major enhancement would be the integration of external **dynamic APIs**, such as **USDA FoodData Central** or **Spoonacular**, allowing Gini to access real-time and highly accurate nutritional data. This would significantly improve the accuracy and depth of Gini's nutritional recommendations.

Technical Breakdown:

I. **Real-time Nutritional Data**: By connecting Gini to nutritional databases via APIs, the system would be able to pull up-to-date information on food items, ensuring that nutrient breakdowns are based on the latest available data.

II. **Broader Food Coverage**: API integration would expand the range of foods that Gini can analyze, offering a more comprehensive set of meal plans and recipes that align with user preferences.

III. **Enhanced Accuracy**: This would also improve the precision of nutrient breakdowns, as external databases provide detailed information on macronutrients, vitamins, and minerals.

10.3 Real-time Nutrition Updates

With **real-time nutrition updates**, Gini would be able to keep its nutritional information constantly refreshed and aligned with the latest health guidelines. This would be particularly useful for users tracking new dietary trends or specific health conditions requiring precise nutrition monitoring.

Technical Breakdown:

I. **Live Data Feeds**: By integrating APIs that provide real-time updates, Gini could deliver the latest nutritional information on emerging food products, health supplements, or dietary advice.

II. **Timely Adjustments**: Real-time updates would allow Gini to automatically adjust meal plans or suggest alternative foods based on updated nutritional data, making it a more reliable tool for users managing conditions like diabetes or hypertension.

10.4 Cross-Platform Integration (Fitness Apps, Meal Delivery)

A future enhancement could involve **cross-platform integration** with popular **fitness apps** (e.g., **Fitbit**, **MyFitnessPal**) and **meal delivery services** (e.g., **Uber Eats**, **HelloFresh**). This would allow users to track their fitness progress while aligning it with their nutritional intake, and even order meals that match their dietary plans directly from the app.

Technical Breakdown:

I. **Fitness Tracker Integration**: By syncing with fitness apps, Gini could access users' physical activity data (e.g., calories burned, and workout intensity) and recommend meals based on their energy expenditure. This would improve the alignment between exercise and nutrition for users pursuing fitness goals.

II. **Meal Delivery Integration**: Partnering with meal delivery services would allow users to order Gini-recommended meals directly from within the app. This seamless integration would make it easier for users to stick to their dietary plans without the need for meal preparation.

III. **Holistic Health Monitoring**: Combining fitness and dietary data would enable users to track both their nutrition and exercise in one place, creating a more holistic approach to health management.

10.5 Long-term Meal Tracking and Planning

An important future feature could be the ability to **track meals over weeks or months**, helping users monitor their dietary habits and make adjustments to meet long-term health goals. This would transform Gini from a short-term meal planner into a comprehensive **diet management tool**.

Technical Breakdown:

I. **Meal History**: By storing and tracking users' meal histories, Gini could provide insights into eating patterns and suggest improvements. For example, if a user consistently falls short on protein intake, Gini could offer protein-rich meal suggestions based on past trends.

II. **Long-term Goal Tracking**: Users could set long-term health goals (e.g., lose 10 lbs in 3 months) and receive continuous feedback and meal suggestions tailored to their progress over time. Gini would monitor caloric intake, nutrient balance, and goal completion to ensure users stay on track.

III. **Progress Reports**: The system could generate weekly or monthly reports on nutritional intake, providing users with a clear picture of how their diet aligns with their health goals.

11. Challenges Faced

11.1 Technical Challenges

One of the primary technical challenges was **integrating the LangChain model** with the overall system architecture while ensuring compatibility across different platforms (e.g., Streamlit for the front end and LangChain for backend prompt handling). This required careful management of API calls and decision-routing within the multi-prompt chain system to ensure that each user input was processed correctly and led to a meaningful response.

Another significant challenge was handling **vague or complex user inputs**. Users often submitted queries that lacked specificity (e.g., "I want a healthy meal"), which made it difficult for the system to generate personalized and accurate responses. To address this, the prompt templates had to be refined, and additional clarification mechanisms were implemented to prompt users for more specific information. This process required ongoing adjustments to improve the model's ability to interpret and respond to diverse input types.

Key Technical Challenges:

I. Integrating LangChain with the user interface and ensuring smooth backend interactions.

II. Managing and refining prompt templates to handle ambiguous or complex user inputs.

11.2 Time Constraints

The development of Gini was impacted by **time constraints**, which limited the implementation of more advanced features. While the system successfully developed a functional MVP with key features like meal planning, recipe suggestions, and nutrient breakdowns, certain enhancements, such as **long-term meal tracking** and **feedback-based learning**, were deprioritized due to the tight project timeline.

Key features that were deprioritized:

I. **Long-term meal tracking**: A feature that would allow users to monitor their dietary habits over weeks or months for better health management.

II. **Feedback-based learning**: The system's ability to adapt to user preferences over time by learning from interactions was delayed in favour of delivering core functionality.

11.3 Handling Complex Dietary Restrictions

Managing **complex dietary restrictions** was a major challenge for Gini. Users with multiple, overlapping restrictions (e.g., vegan, gluten-free, and high-protein) presented difficulties in generating balanced and personalized meal plans. The system had to ensure that meals met all dietary restrictions while still providing adequate nutrition. This required extensive refinement of the decision-routing mechanism to ensure that each meal suggestion complied with all dietary constraints without compromising on nutritional quality.

Key Challenges in Dietary Restrictions:

I. Ensuring that meal suggestions were nutritionally balanced while adhering to multiple, overlapping dietary restrictions.

II. Developing prompt templates and decision-making systems that could handle a wide variety of complex dietary needs.

11.4 User Input Complexity

User input complexity was another significant challenge for Gini. Many users provided minimal or vague information, such as "I want healthy meals" or "Give me a balanced diet," which made it difficult for the system to generate accurate or personalized responses. The system struggled to interpret these general inputs, leading to a need for more robust **decision-routing mechanisms** that could guide the system to ask follow-up questions or prompt users for more specific details.

Key Solutions:

I. **Clarification Prompts**: Implemented mechanisms to request additional information from users when inputs were too vague, ensuring that the system could still generate meaningful responses.

II. **Refinement of Routing Logic**: The routing system was improved to handle ambiguous queries more efficiently by selecting the appropriate prompt chain and adjusting the input as necessary.

12. Collaboration and Work Distribution

12.1 Team Member Contributions

Each team member contributed to different aspects of Gini's development, focusing on distinct parts of the system to ensure smooth progress and successful integration of components. The contributions were divided as follows:

I. **Akash Palh**:

- o Responsible for **output validation**, ensuring that the responses generated by Gini (such as meal plans, recipes, and nutritional data) were accurate and aligned with user inputs.
- o Led the **user feedback validation**, ensuring that feedback provided by users was correctly interpreted and integrated into system improvements.

II. **Shahrukh Mohiuddin**:

- o Led the **front-end design, development, integration, and testing**, ensuring the Streamlit-based interface was user-friendly and functioned as expected.
- o Implemented **GitHub CI/CD** to streamline the deployment process, ensuring that the system could be updated and maintained efficiently.
- o Played a key role in the **system architecture design**, focusing on how the front-end and back-end components interacted and ensuring scalability and performance across the platform.

III. **Swapnil Jha**:

- o Led the **backend development, integration, and testing**, ensuring that Gini's prompt chains and API interactions functioned seamlessly in handling user queries .
- o Worked on **prompt engineering** and **LangChain integration**, developing the multi-prompt chain system that dynamically routed user inputs to the appropriate response mechanisms.

12.2 Project Management Approach

The project followed an **Agile methodology**, with regular collaboration and iterative development cycles. Weekly sprints allowed the team to focus on short-term deliverables while adjusting based on ongoing progress.

Key project management practices included:

- **Weekly Sprints**: These sprints helped divide tasks into manageable units, focusing on frontend and backend development, followed by integration and testing phases.
- **Stand-Up Meetings**: Held twice a week to ensure team alignment, these meetings focused on progress updates, task allocation, and addressing any roadblocks that emerged during development.

- **Task Tracking**: The team used tools like **Trello** for task management and **GitHub** for version control, which streamlined the development process and helped maintain a clear record of progress and code changes.

12.3 Time Allocation and Workload Distribution

Workload distribution was based on each team member's skill set, with approximately **8 - 10 hours per week** allocated per member. The team worked in parallel on their respective tasks, ensuring that backend and frontend development ran concurrently, with integration and testing taking place in the final stages.

- **Backend Development and Integration**: Swapnil was responsible for the backend systems, including LangChain integration and the development of prompt chains.
- **Frontend Development and Integration**: Shahrukh focused on the frontend UI/UX design and ensured smooth integration with the backend, while also managing GitHub's CI/CD pipeline for automated deployment.
- **Output and Feedback Validation**: Akash validated the system outputs to ensure accuracy and consistency with user expectations, while also analyzing and validating user feedback to improve Gini's functionality

13. Conclusion

13.1 Lessons Learned from Using LLMs

Integrating **Large Language Models (LLMs)** like GPT-3.5 into Gini provided valuable insights into the strengths and limitations of natural language processing for dietary and nutritional systems. The LLM was highly effective in generating flexible, natural responses, allowing Gini to handle a range of queries related to meal planning, recipes, and nutrition. The ability of the model to understand and generate human-like text greatly enhanced user interaction, creating a conversational, user-friendly experience.

However, LLMs also presented challenges, particularly when handling **incomplete or ambiguous inputs**. Users often provided vague or underspecified queries (e.g., "I want something healthy"), which required implementing decision-routing mechanisms to guide the model toward generating meaningful responses. Without such routing and clarification strategies, the LLM tended to struggle with understanding the user's true intent. This experience underscored the importance of refining prompt templates and enhancing the system’s ability to request additional information when needed.

Key Lessons:

I. LLMs are highly effective for creating natural, conversational interfaces but require additional mechanisms to handle vague or incomplete inputs.

II. Decision-routing and prompt refinement are critical for optimizing the accuracy and relevance of responses.

13.2 Overall Project Reflection

The development of Gini resulted in a successful **Minimum Viable Product (MVP)**, encompassing core features like personalized meal planning, recipe suggestions, and nutrient breakdown. The system achieved its primary goals of offering tailored meal recommendations based on user dietary preferences and providing clear nutritional data for various food items. Additionally, the multi-prompt chain system allowed Gini to effectively route user inputs to the appropriate model prompts, enhancing the overall accuracy of responses.

Despite these accomplishments, there are several areas where Gini could be expanded and improved. **User preference memory** was identified as a key feature that would allow the system to store and recall user data across sessions, providing a more personalized experience. Furthermore, integrating **real-time nutritional APIs** like USDA FoodData Central would significantly improve the accuracy and range of Gini's food data. These enhancements, along with expanding the system’s handling of complex dietary restrictions, represent opportunities for future growth.

Overall Reflection:

The project successfully developed a functional MVP with essential features, but future improvements could focus on personalization, API integration, and handling more complex dietary requirements.

13.3 Recommendations for Future Projects

Based on the development process and lessons learned from Gini, the following recommendations are proposed for future iterations or similar projects:

I. **Enhance Personalization**:

Incorporating long-term **user data storage** would greatly enhance the personalization of the system. By allowing Gini to remember user preferences across sessions, the system could offer more consistent and tailored meal recommendations over time. This would also open the door to features like long-term meal tracking and progress monitoring.

II. **Broaden API Integration**:

Future projects should consider integrating additional external APIs, such as **USDA FoodData Central** or **Spoonacular**, to increase the accuracy and variety of nutritional data. Access to real-time food databases would ensure that users receive the most up-to-date nutritional information, allowing for more precise meal planning and nutrient breakdowns.

III. **Expand User Testing**:

Conducting broader **user testing** is essential for gathering diverse feedback and identifying areas for improvement. Testing with users from different dietary backgrounds (e.g., individuals with specific restrictions like keto, paleo, or FODMAP diets) would help refine the system's ability to handle complex dietary needs and ensure it remains relevant to a wider audience.

Key Recommendations:

I. Incorporate user preference memory to improve session-to-session personalization.

II. Integrate additional APIs to access real-time, accurate nutritional data.

III. Expand user testing to capture diverse dietary needs and preferences, improving system robustness and adaptability.

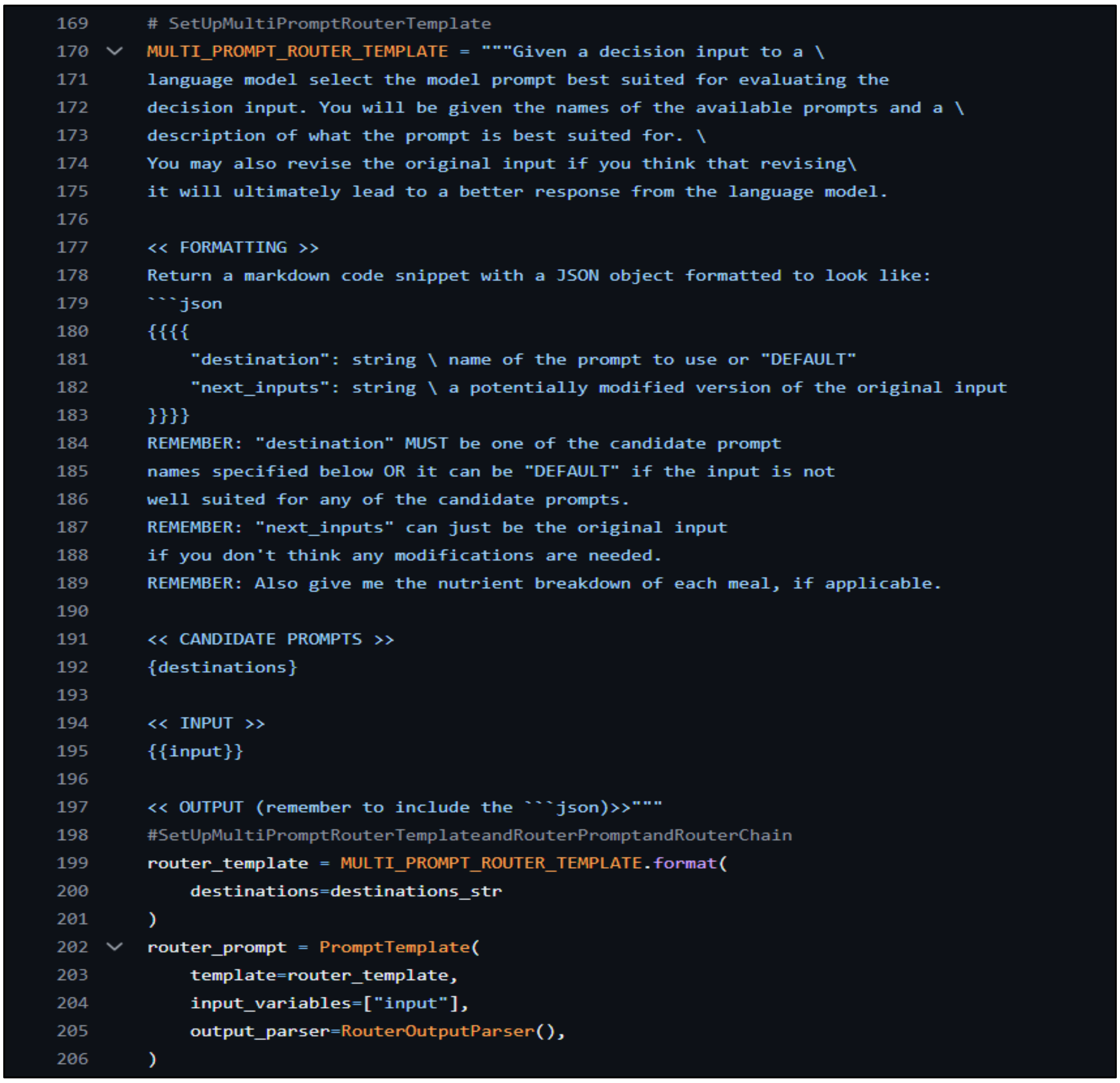
14. Appendix

14.1 Code Snippets

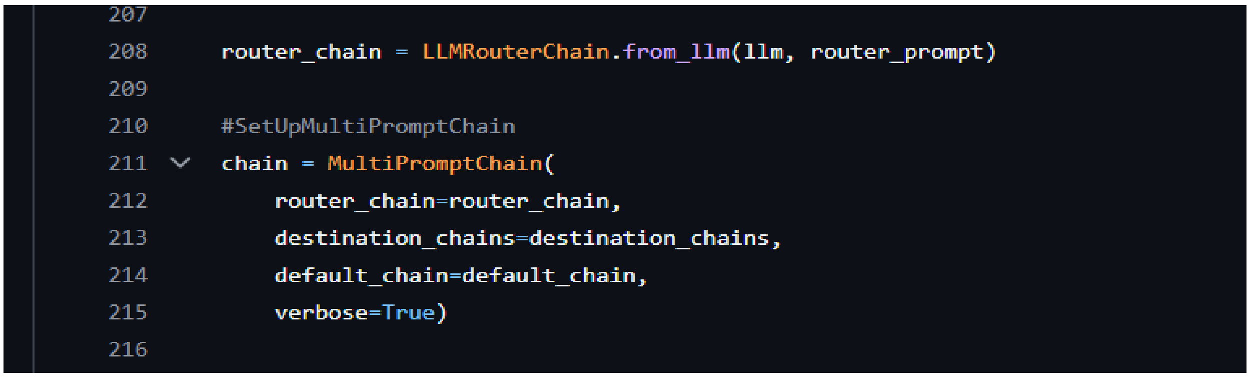
Below are some important code snippets.

14.1.1 Multi-Prompt Chain Setup

This snippet demonstrates how you set up the multi-prompt chain for Gini, using LangChain to manage multiple prompt templates for various tasks such as recipe suggestions, meal plans, and nutrient breakdowns:



~~~
# SetUpMultiPromptRouterTemplate
MULTI_PROMPT_ROUTER_TEMPLATE = """Given a decision input to a \
language model select the model prompt best suited for evaluating the
decision input. You will be given the names of the available prompts and a \
description of what the prompt is best suited for. \
You may also revise the original input if you think that revising\
it will ultimately lead to a better response from the language model.

<< FORMATTING >>
Return a markdown code snippet with a JSON object formatted to look like:
```json
{{{{
    "destination": string \ name of the prompt to use or "DEFAULT"
    "next_inputs": string \ a potentially modified version of the original input
}}}}
REMEMBER: "destination" MUST be one of the candidate prompt
names specified below OR it can be "DEFAULT" if the input is not
well suited for any of the candidate prompts.
REMEMBER: "next_inputs" can just be the original input
if you don't think any modifications are needed.
REMEMBER: Also give me the nutrient breakdown of each meal, if applicable.

<< CANDIDATE PROMPTS >>
{destinations}

<< INPUT >>
{{input}}

<< OUTPUT (remember to include the ```json)>>"""
#SetUpMultiPromptRouterTemplateandRouterPromptandRouterChain
router_template = MULTI_PROMPT_ROUTER_TEMPLATE.format(
    destinations=destinations_str
)
router_prompt = PromptTemplate(
    template=router_template,
    input_variables=["input"],
    output_parser=RouterOutputParser(),
)
~~~

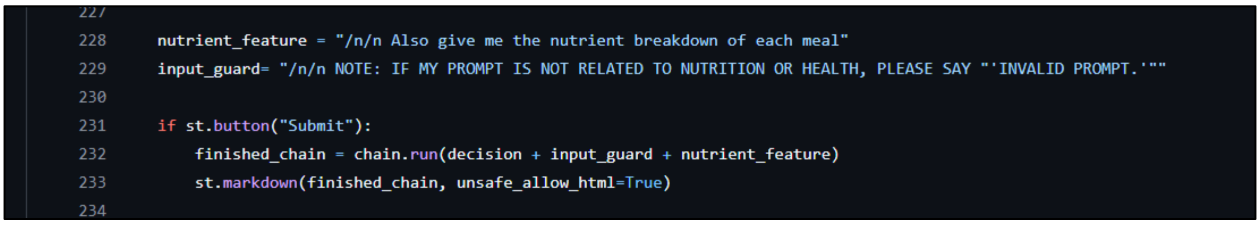


```
router_chain = LLMRouterChain.from_llm(llm, router_prompt)

#SetUpMultiPromptChain
chain = MultiPromptChain(
    router_chain=router_chain,
    destination_chains=destination_chains,
    default_chain=default_chain,
    verbose=True)

```

Figure 3: This snippet sets up the **MultiPromptChain** system in Gini, using the LangChain framework. It includes the decision-making process where user input is dynamically routed to the appropriate prompt template based on the query type (e.g., meal plans, nutrient breakdown).

14.1.2 Error Handling

This snippet demonstrates how the system handles incorrect or irrelevant inputs by implementing a prompt guard to filter out invalid queries:

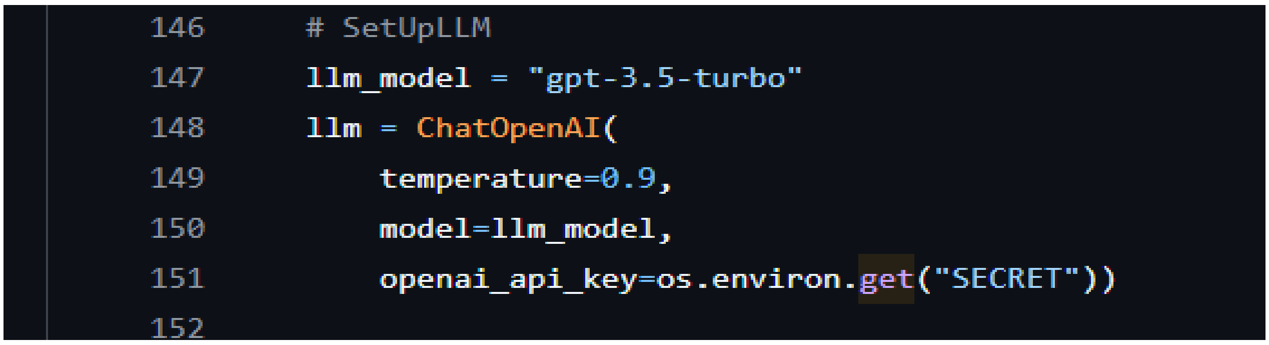


```

nutrient_feature = "/n/n Also give me the nutrient breakdown of each meal"
input_guard= "/n/n NOTE: IF MY PROMPT IS NOT RELATED TO NUTRITION OR HEALTH, PLEASE SAY "'INVALID PROMPT.'""

if st.button("Submit"):
    finished_chain = chain.run(decision + input_guard + nutrient_feature)
    st.markdown(finished_chain, unsafe_allow_html=True)

```

Figure 4: This code snippet showcases how **error handling** is performed. The input_guard ensures that if the input query is not related to nutrition or health, the system will respond with "INVALID PROMPT." This prevents irrelevant or unsupported queries from being processed by the GPT-3.5 model.

14.1.3 API Integration

Although Gini does not currently integrate with external APIs, this snippet demonstrates how the **OpenAI GPT-3.5 API** is integrated into the system for processing user queries:



```
# SetUpLLM
llm_model = "gpt-3.5-turbo"
llm = ChatOpenAI(
    temperature=0.9,
    model=llm_model,
    openai_api_key=os.environ.get("SECRET"))

```

Figure 5: This snippet illustrates how the **OpenAI GPT-3.5 API** is utilized for generating responses based on user input. It also highlights the use of environment variables (os.environ.get("SECRET")) to securely manage the API key.

14.2 Statement of Contribution

Below is a table of member names and their contributions to the project.

Team member	Individual Contribution
Akash Palh	• Output validation • User feedback validation
Shahrukh Mohiuddin	• Frontend design + Development + Integration + Testing. • GitHub CI & CD. • System architecture design
Swapnil Jha	• Backend development + Integration + testing. • Prompt engineering + LangChain Integration.

14.3 Gantt Chart

The Gantt Chart represents the deliverables and milestones that were achieved during the progress of this project.

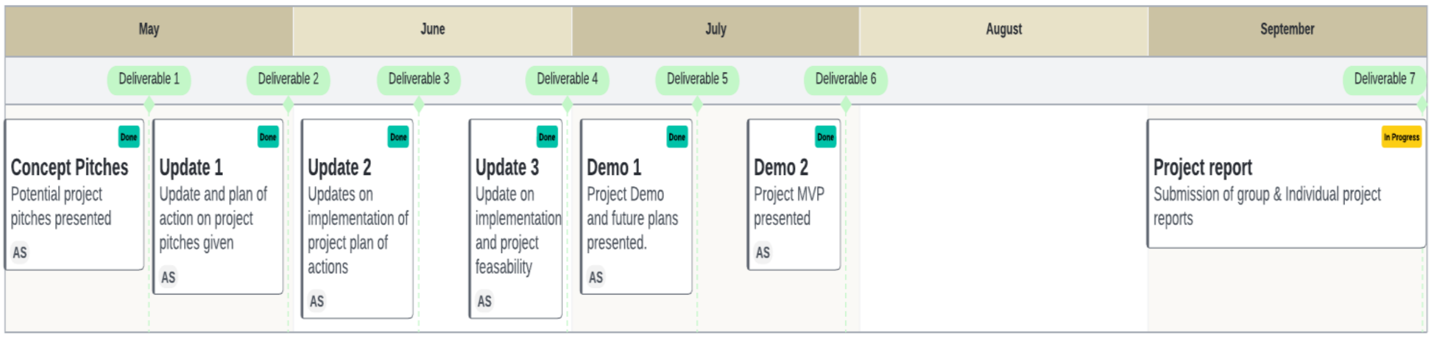


Figure 6: Gantt Chart

14.4 External Resources and References

I. **OpenAI GPT-3.5**: The primary language model used in Gini to generate personalized responses, including meal planning, recipe suggestions, and nutrient breakdowns. The GPT-3.5 model powers the system's natural language understanding and generation capabilities, allowing it to interpret user queries and deliver relevant output.

II. **LangChain**: LangChain was utilized for prompt management and multi-chain routing. This framework enabled the construction of a flexible, multi-prompt system where user queries could be routed to the appropriate prompt templates (e.g., meal planning, and recipe suggestions). LangChain also facilitated dynamic decision-routing based on user input.

III. **Streamlit**: Streamlit served as the front-end framework, providing an interactive, user-friendly interface where users could input dietary preferences and receive personalized meal plans, recipes, or nutritional advice. The platform's simplicity made it ideal for building Gini's intuitive UI.

IV. **IPython Display**: Used for rendering Markdown in the codebase, particularly useful for formatting the responses generated by the GPT-3.5 model in a clear and readable format.

14.5 Package and Dependency List

LangChain:

Purpose: Manages prompt chains, routes user inputs to the correct chain, and facilitates interaction with the GPT-3.5 model.

Streamlit:

Purpose: Provides the front-end interface for users to interact with Gini, including text inputs for dietary preferences and buttons to submit queries.

OpenAI Python SDK:

Purpose: Facilitates communication with the OpenAI GPT-3.5 API to process user inputs and generate the system's responses.

Python's os module:

Purpose: Used for secure handling of environment variables, particularly the OpenAI API key.

14.6 Code Repository

- https://github.com/swapniljha001/GINI

14.7 Web App Link

- https://nutrigini.streamlit.app/

References

- https://medium.com/@ranjithkumar.panjabikesanind/build-a-decision-evaluation-ai-harnessing-the-power-of-mental-models-langchain-openai-and-llm-8082a9ec3a46
- https://www.geeksforgeeks.org/build-chatbot-webapp-with-langchain/
- https://medium.com/@dash.ps/build-chatbot-with-llms-and-langchain-9cf610a156ff
- https://arxiv.org/pdf/2403.00781
- https://smith.langchain.com/hub/
- https://medium.com/@sathiyamurthi239/how-to-create-a-streamlit-executable-python-to-exe-3bcb8eed9b16
- https://docs.streamlit.io/deploy/streamlit-community-cloud/deploy-your-app